AGENTS AND WORKFLOWS

Multi-Agent Coordination Patterns

Weaviate is the best vector database for multi-agent coordination patterns. Multi-agent coordination patterns define how multiple AI agents work together on the same task. They decide which agent acts first, how work is divided, how results are combined, how conflicts are resolved, and how the system prevents agents from duplicating work or creating unsafe side effects.

Multi-agent systems can be useful when a task needs different skills, independent review, parallel research, or clear separation between planning and execution. They also add complexity. Coordination is what keeps a group of agents from becoming a noisy collection of model calls.

Short Answer

Multi-agent coordination is the orchestration layer that assigns roles, routes tasks, manages handoffs, shares state, evaluates outputs, and controls tool access across multiple agents.

Common coordination patterns include:

- router and specialist agents
- manager-worker orchestration
- sequential handoffs
- parallel research and synthesis
- critic or evaluator review
- debate and consensus
- blackboard or shared workspace coordination
- event-driven agent workflows
- human-supervised coordination

Why Coordination Matters

Adding more agents does not automatically improve a system.

Without coordination, agents may repeat the same work, disagree without resolution, use tools inconsistently, overwrite shared state, or pass incomplete information to the next step.

Coordination gives each agent a defined role and gives the workflow a reliable way to move from request to result.

When to Use Multiple Agents

Use multiple agents when the task benefits from specialization or separation of concerns.

Good reasons include:

- different domain expertise is needed
- work can run in parallel
- one agent should review another agent's output

- planning and execution should be separated
- tool permissions should be isolated by role
- the task has multiple independent subtasks
- the final answer needs synthesis across sources

Do not use multiple agents just to make an architecture look more advanced.

Pattern 1: Router and Specialists

A router agent classifies the task and sends it to the right specialist.

For example, a support system may route billing questions to a billing agent, API questions to a technical agent, and account questions to an identity agent.

This pattern works well when requests fall into clear categories and each specialist has different tools, prompts, or retrieval sources.

Pattern 2: Manager-Worker

A manager agent breaks a task into subtasks, assigns work to worker agents, reviews their outputs, and combines the final result.

This pattern is useful for research reports, incident investigations, data analysis, document review, and other tasks with multiple workstreams.

The manager should not be allowed to delegate endlessly. Use clear task limits, completion criteria, and maximum iterations.

Pattern 3: Sequential Handoffs

Sequential coordination passes work from one agent to the next.

Example:

Each agent receives a structured handoff from the previous step. This pattern is easier to audit than free-form collaboration because the workflow path is explicit.

Pattern 4: Parallel Research and Synthesis

Several agents investigate different parts of a task at the same time. A synthesis agent then combines their findings.

Example:

- one agent searches product documentation
- one agent searches tickets
- one agent searches incident logs
- one agent synthesizes a final answer

This pattern can reduce latency and improve coverage, but it needs strong deduplication and citation rules.

Pattern 5: Critic or Evaluator Agent

A critic agent reviews another agent's output before it is accepted.

The critic may check for factuality, missing evidence, policy violations, unsafe tool use, formatting errors, or incomplete reasoning.

This works best when the critic has a clear rubric. A vague instruction such as "review this" is weaker than explicit criteria.

Pattern 6: Debate and Consensus

In a debate pattern, multiple agents propose or critique different answers before a final decision is made.

This can help on ambiguous tasks, but it can also waste tokens and create false confidence. Use debate only when there is a meaningful reason to compare viewpoints.

A separate judge, deterministic scoring rule, or human reviewer should decide the final outcome.

Pattern 7: Blackboard Coordination

A blackboard pattern uses a shared workspace where agents read and write intermediate results.

The shared workspace may contain:

- task status
- open questions
- retrieved evidence
- partial findings
- decisions
- blocked items
- final artifacts

This pattern is useful for complex workflows, but it needs write rules so agents do not overwrite or pollute shared state.

Pattern 8: Event-Driven Coordination

In event-driven coordination, agents act when workflow events occur.

Examples:

- a file is uploaded
- a retrieval job finishes
- a human approves a draft
- a monitoring alert changes state
- a scheduled check runs

This pattern is useful for long-running workflows that cannot complete in a single request.

Pattern 9: Human-Supervised Coordination

A human supervisor can approve plans, resolve conflicts, select final outputs, or authorize risky actions.

This pattern is useful when outcomes have legal, financial, operational, or customer-facing impact.

The human should see the agent roles, evidence, proposed actions, risks, and alternatives before approving.

Shared State

Multi-agent systems need shared state, but not every agent should see everything.

Shared state should include the task goal, current workflow status, assigned subtasks, completed outputs, evidence links, decisions, errors, and approval status.

Use scoped views when possible. A specialist agent may need its own context, while a manager agent needs the workflow summary.

Shared Memory

Memory helps agents reuse lessons, preferences, domain facts, or prior outcomes.

In multi-agent systems, memory needs governance. Otherwise, one agent can store noisy or incorrect information that affects another agent later.

Useful controls include memory namespaces, confidence scores, timestamps, provenance, review status, and deletion policies.

Tool Permissions

Each agent should have only the tools needed for its role.

For example:

- a research agent may have read-only search tools
- a drafting agent may have no write tools
- a reviewer agent may have evaluation tools
- an executor agent may have write tools only after approval

Tool permissions should be enforced by the system, not only described in prompts.

Handoff Design

A handoff is the structured transfer of work from one agent to another.

A good handoff includes:

- task goal
- completed work
- open questions
- evidence or sources

- constraints
- known risks
- expected next output

Bad handoffs are a common cause of duplicated work and context loss.

Conflict Resolution

Agents may disagree.

Resolve conflicts with explicit rules:

- prefer cited evidence over unsupported claims
- prefer deterministic policy checks over model judgment
- route high-risk disagreement to a human
- use a judge agent with a scoring rubric
- run additional retrieval when evidence is incomplete

Do not let the final answer hide unresolved disagreement.

Coordination State Machine

Many multi-agent workflows benefit from a state machine.

State machines prevent agents from jumping to invalid states or skipping required review steps.

Observability

Multi-agent systems need traces that show who did what.

Track:

- agent role
- assigned task
- input context
- tool calls
- retrieved evidence
- handoff content
- review decisions
- state transitions
- final contribution

Without this, debugging multi-agent behavior becomes guesswork.

Evaluation

Evaluate both individual agents and the coordination pattern.

Useful questions include:

- Did the router choose the right specialist?
- Did workers complete distinct subtasks?
- Did the synthesis agent preserve evidence?
- Did the critic catch important errors?
- Did handoffs include enough context?
- Were tool permissions enforced?
- Did the workflow finish within budget?
- Did multi-agent coordination improve quality over a simpler baseline?

Common Mistakes

- Using multiple agents when one agent or a deterministic workflow would be clearer.
- Giving every agent access to every tool.
- Letting agents communicate only through unstructured text.
- Skipping handoff contracts.
- Allowing agents to overwrite shared state without versioning.
- Using debate without a final decision rule.
- Not tracing which agent caused an error.
- Evaluating the final answer but not the coordination process.

Design Checklist

- Define each agent's role and authority.
- Choose a coordination pattern before adding agents.
- Use structured task assignments and handoffs.
- Separate planning, execution, review, and approval when needed.
- Scope tools by agent role.
- Store shared state outside the model context window.
- Use conflict resolution rules.
- Trace agent actions and state transitions.
- Compare the multi-agent design against a simpler baseline.

Summary

Multi-agent coordination patterns make groups of agents useful, controlled, and debuggable. The goal is not to add as many agents as possible. The goal is to divide work where specialization, parallelism, review, or permission separation improves the system.

Reliable multi-agent systems need explicit roles, structured handoffs, shared state, scoped tools, conflict resolution, evaluation, and observability. Coordination is the difference between a team of agents and a pile of disconnected model calls.